Ensembel Models: Bagging and Random Forests

Justin Post

Recap

Looked at a few common supervised learning models for regression and classification tasks

- MLR, Penalized MLR, & Regression Trees
- Logistic Regression & Classification Trees

Now we'll investigate commonly used *ensemble* methods

Prediction with Tree Based Methods

If we care mostly about prediction not interpretation

- Often use **bootstrapping** to get multiple samples to fit on
- Can average across many fitted trees
- Decreases variance over an individual tree fit

Prediction with Tree Based Methods

If we care mostly about prediction not interpretation

- Often use **bootstrapping** to get multiple samples to fit on
- Can average across many fitted trees
- Decreases variance over an individual tree fit

Major ensemble tree methods

1. Bagging (boostrap aggregation)
2. Random Forests (extends idea of bagging - includes bagging as a special case)
3. Boosting (*slow* training of trees)

Bagging

Bagging = Bootstrap Aggregation - a general method

Bootstrapping

- resample from the data (non-parametric) or a fitted model (parameteric)
- for non-parameteric
  - treats sample as population
  - resampling done with replacement
  - can get same observation multiple times

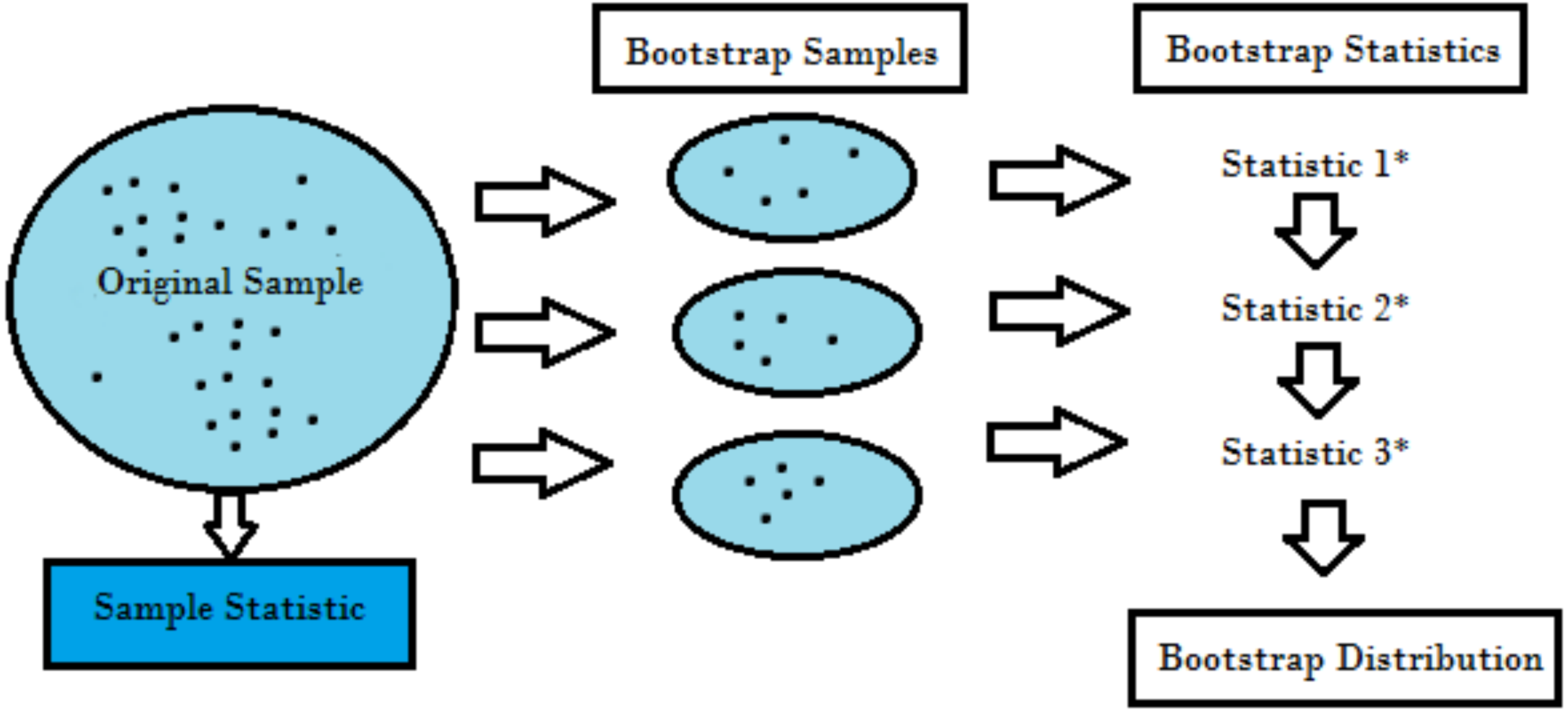
Bagging

Bagging = Bootstrap Aggregation - a general method

Bootstrapping

- resample from the data (non-parametric) or a fitted model (parameteric)
- for non-parameteric
    - treats sample as population
    - resampling done with replacement
    - can get same observation multiple times
- method or estimation applied to each resample
- traditionally used to obtain standard errors (measures of variability) or construct confidence intervals

Non-Parametric Bootstrapping



Bagging

- Create many bootstrap (re)samples, $j = 1, \ldots, B$

Bagging

- Create many bootstrap (re)samples, $j = 1, \ldots, B$
- Fit tress to each (re)sample
    - Have B fitted trees

Bagging

- Create many bootstrap (re)samples, $j = 1, \ldots, B$
- Fit tress to each (re)sample
  - Have B fitted trees
- For a given set of predictor values, find $\hat{y}$ for each tree
  - Call prediction for a given set of x values $\hat{y}^{*j}(x)$

Bagging

- Create many bootstrap (re)samples, $j = 1, \ldots, B$
- Fit tress to each (re)sample
  - Have B fitted trees
- For a given set of predictor values, find $\hat{y}$ for each tree
  - Call prediction for a given set of x values $\hat{y}^{*j}(x)$
- Combine the predictions from the trees to create the final prediction!
  - For regression trees, usually use the average of the predictions

$$\hat{y}(x) = \frac{1}{B}\sum_{j=1}^{B} \hat{y}^{*j}(x)$$

Bagging

- Create many bootstrap (re)samples, $j = 1, \ldots, B$
- Fit tress to each (re)sample
  - Have B fitted trees
- For a given set of predictor values, find $\hat{y}$ for each tree
  - Call prediction for a given set of x values $\hat{y}^{*j}(x)$
- Combine the predictions from the trees to create the final prediction!
  - For classification trees, usually use the **majority vote**

$$\text{Use most common prediction made by all bootstrap trees}$$

Using `tidymodels` to Fit a Bagged Tree Model

- Read in our heart disease data

```r
library(tidyverse)
library(tidymodels)
heart_data <- read_csv("https://www4.stat.ncsu.edu/online/datasets/heart.csv") |>
  filter(RestingBP > 0) #remove one value
heart_data <- heart_data |> mutate(HeartDisease = factor(HeartDisease))
heart_split <- initial_split(heart_data, prop = 0.8)
heart_train <- training(heart_split)
heart_test <- testing(heart_split)
heart_CV_folds <- vfold_cv(heart_train, 10)
heart_data
```

```
## # A tibble: 917 x 12
##     Age Sex   ChestPainType RestingBP Cholesterol FastingBS RestingECG MaxHR
##   <dbl> <chr> <chr>             <dbl>       <dbl>     <dbl> <chr>      <dbl>
## 1    40 M     ATA                 140         289         0 Normal       172
## 2    49 F     NAP                 160         180         0 Normal       156
## 3    37 M     ATA                 130         283         0 ST            98
## 4    48 F     ASY                 138         214         0 Normal       108
## 5    54 M     NAP                 150         195         0 Normal       122
## # i 912 more rows
## # i 4 more variables: ExerciseAngina <chr>, Oldpeak <dbl>, ST_Slope <chr>,
## #   HeartDisease <fct>
```

Using `tidymodels` to Fit a Bagged Tree Model

- Recall: For tree based methods we don't need to worry about interactions
- Can reuse the recipes from previous!

```
LR3_rec <- recipe(HeartDisease ~ Age + Sex + ChestPainType + RestingBP + RestingECG + MaxHR + ExerciseAngina,
                  data = heart_train) |>
  step_normalize(all_numeric(), -HeartDisease) |>
  step_dummy(Sex, ChestPainType, RestingECG, ExerciseAngina)
LR3_rec |>
  prep(heart_train) |>
  bake(heart_train) |>
  colnames()
```

```
##  [1] "Age"               "RestingBP"         "MaxHR"
##  [4] "HeartDisease"      "Sex_M"             "ChestPainType_ATA"
##  [7] "ChestPainType_NAP" "ChestPainType_TA"  "RestingECG_Normal"
## [10] "RestingECG_ST"     "ExerciseAngina_Y"
```

Using `tidymodels` to Fit a Bagged Tree Model

- Now set up our model type and engine
- Using this parsnip model
    - Could tune on a few things here if we'd like

```
bag_spec <- bag_tree(tree_depth = 5, min_n = 10, cost_complexity = tune()) |>
  set_engine("rpart") |>
  set_mode("classification")
```

Using `tidymodels` to Fit a Bagged Tree Model

- Create our workflows

```
#install baguette package if not already done!
library(baguette)
bag_wkf <- workflow() |>
  add_recipe(LR3_rec) |>
  add_model(bag_spec)
```

Using `tidymodels` to Fit a Bagged Tree Model

- Fit to our CV folds!
    - Note: CV isn't really necessary. We could use out-of-bag observations to determine how well our model works instead!

```
bag_fit <- bag_wkf |>
  tune_grid(resamples = heart_CV_folds,
            grid = grid_regular(cost_complexity(),
                                levels = 15),
            metrics = metric_set(accuracy, mn_log_loss))
bag_fit
```

```
## # Tuning results
## # 10-fold cross-validation
## # A tibble: 10 x 4
##    splits            id     .metrics          .notes
##    <list>            <chr>  <list>            <list>
##  1 <split [659/74]> Fold01 <tibble [30 x 5]> <tibble [0 x 3]>
##  2 <split [659/74]> Fold02 <tibble [30 x 5]> <tibble [0 x 3]>
##  3 <split [659/74]> Fold03 <tibble [30 x 5]> <tibble [0 x 3]>
##  4 <split [660/73]> Fold04 <tibble [30 x 5]> <tibble [0 x 3]>
##  5 <split [660/73]> Fold05 <tibble [30 x 5]> <tibble [0 x 3]>
##  6 <split [660/73]> Fold06 <tibble [30 x 5]> <tibble [0 x 3]>
##  7 <split [660/73]> Fold07 <tibble [30 x 5]> <tibble [0 x 3]>
##  8 <split [660/73]> Fold08 <tibble [30 x 5]> <tibble [0 x 3]>
##  9 <split [660/73]> Fold09 <tibble [30 x 5]> <tibble [0 x 3]>
```

Using `tidymodels` to Fit a Bagged Tree Model

- Check our metrics across the folds!
- Look at log loss and sort it

```
bag_fit |>
  collect_metrics() |>
  filter(.metric == "mn_log_loss") |>
  arrange(mean)
```

```
## # A tibble: 15 x 7
##    cost_complexity .metric     .estimator  mean     n std_err .config
##              <dbl> <chr>       <chr>      <dbl> <int>   <dbl> <chr>
##  1        3.16e- 6 mn_log_loss binary     0.452    10  0.0201 Preprocessor1_Mod~
##  2        1.39e- 5 mn_log_loss binary     0.455    10  0.0177 Preprocessor1_Mod~
##  3        3.73e- 8 mn_log_loss binary     0.455    10  0.0171 Preprocessor1_Mod~
##  4        1.64e- 7 mn_log_loss binary     0.455    10  0.0163 Preprocessor1_Mod~
##  5        4.39e-10 mn_log_loss binary     0.456    10  0.0178 Preprocessor1_Mod~
##  6        1   e-10 mn_log_loss binary     0.457    10  0.0162 Preprocessor1_Mod~
##  7        1.18e- 3 mn_log_loss binary     0.458    10  0.0178 Preprocessor1_Mod~
##  8        5.18e- 3 mn_log_loss binary     0.458    10  0.0138 Preprocessor1_Mod~
##  9        2.68e- 4 mn_log_loss binary     0.459    10  0.0184 Preprocessor1_Mod~
## 10        6.11e- 5 mn_log_loss binary     0.461    10  0.0180 Preprocessor1_Mod~
## 11        7.20e- 7 mn_log_loss binary     0.466    10  0.0186 Preprocessor1_Mod~
## 12        8.48e- 9 mn_log_loss binary     0.469    10  0.0173 Preprocessor1_Mod~
## 13        1.93e- 9 mn_log_loss binary     0.472    10  0.0170 Preprocessor1_Mod~
## 14        2.28e- 2 mn_log_loss binary     0.482    10  0.0119 Preprocessor1_Mod~
```

Using `tidymodels` to Fit a Bagged Tree Model

- Get our best tuning parameter

```
bag_best_params <- select_best(bag_fit, metric = "mn_log_loss")
bag_best_params
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <chr>
## 1      0.00000316 Preprocessor1_Model08
```

- Refit on the entire training set using this tuning parameter

```
bag_final_wkf <- bag_wkf |>
  finalize_workflow(bag_best_params)
bag_final_fit <- bag_final_wkf |>
  last_fit(heart_split, metrics = metric_set(accuracy, mn_log_loss))
```

Using `tidymodels` to Fit a Logistic Regression Model

- For comparison, let's fit our same logistic regression model from previous

```r
LR_spec <- logistic_reg() |>
  set_engine("glm")
LR3_wkf <- workflow() |>
  add_recipe(LR3_rec) |>
  add_model(LR_spec)
LR3_fit <- LR3_wkf |>
  fit_resamples(heart_CV_folds, metrics = metric_set(accuracy, mn_log_loss))
rbind(LR3_fit |> collect_metrics(),
      bag_fit |> collect_metrics() |>
        filter(cost_complexity == bag_best_params$cost_complexity) |>
        select(-cost_complexity))
```

```
## # A tibble: 4 x 6
##   .metric     .estimator  mean     n std_err .config
##   <chr>       <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy    binary     0.782    10  0.0121 Preprocessor1_Model1
## 2 mn_log_loss binary     0.447    10  0.0163 Preprocessor1_Model1
## 3 accuracy    binary     0.795    10  0.0129 Preprocessor1_Model08
## 4 mn_log_loss binary     0.452    10  0.0201 Preprocessor1_Model08
```

Using `tidymodels` to Compare Our Models

- Take these models to the test set and see how they do

```
#test on the test set!
LR_final_fit <- LR3_wkf |>
  last_fit(heart_split, metrics = metric_set(accuracy, mn_log_loss))

LR_final_fit |> collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.826 Preprocessor1_Model1
## 2 mn_log_loss binary         0.442 Preprocessor1_Model1
```

```
bag_final_fit |> collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.810 Preprocessor1_Model1
## 2 mn_log_loss binary         0.495 Preprocessor1_Model1
```

Investigate the Bagged Tree Model

- As before, we can extract our final model and check it out
- Let's first refit to the entire data set

```
bag_full_fit <- bag_final_wkf |>
  fit(heart_data)
bag_full_fit
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: bag_tree()
##
## -- Preprocessor ----------------------------------------------------------------
## 2 Recipe Steps
##
## * step_normalize()
## * step_dummy()
##
## -- Model -----------------------------------------------------------------------
## Bagged CART (classification with 11 members)
##
## Variable importance scores include:
##
## # A tibble: 10 x 4
##    term                value std.error  used
##    <chr>               <dbl>     <dbl> <int>
##  1 ExerciseAngina_Y   116         4.25    11
```

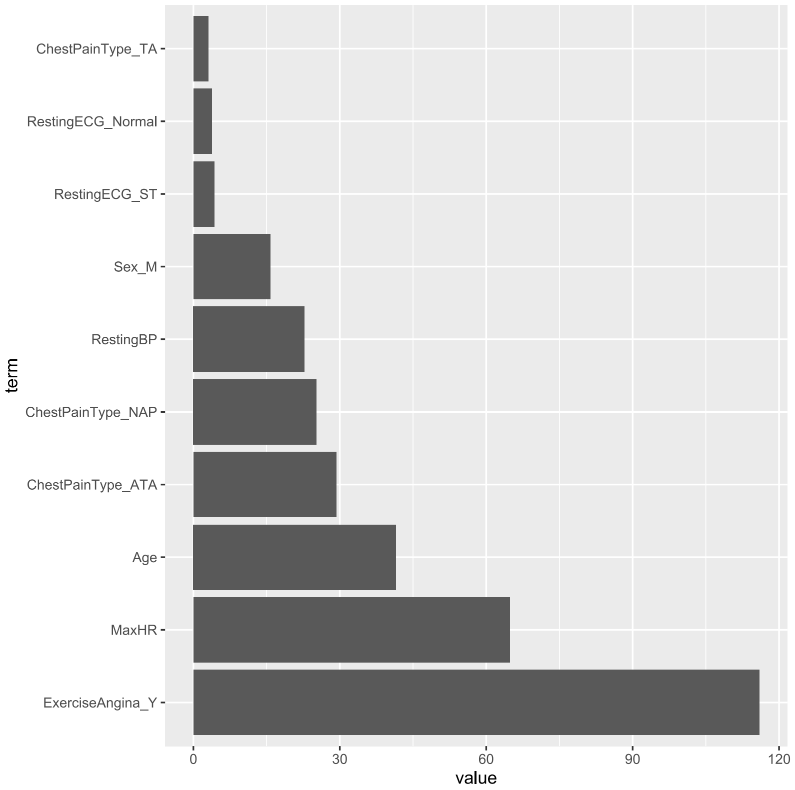
Investigate the Bagged Tree Model

- As before, we can extract our final model and check it out
- Extract the final model and then plot the variable importance

```
bag_final_model <- extract_fit_engine(bag_full_fit)
bag_final_model$imp |>
  mutate(term = factor(term, levels = term)) |>
  ggplot(aes(x = term, y = value)) +
  geom_bar(stat ="identity") +
  coord_flip()
```

Investigate the Bagged Tree Model

- As before, we can extract our final model and check it out
- Extract the final model and then plot the variable importance



Random Forests

- Uses same idea as bagging
- Create multiple trees from bootstrap samples
- Average results in some way for final prediction

Difference:

- Doesn't use all predictors at each step!
- Considers splits using a random subset of predictors each time (# is a tuning parameter)

Random Forests

- Uses same idea as bagging
- Create multiple trees from bootstrap samples
- Average results in some way for final prediction

Difference:

- Doesn't use all predictors at each step!
- Considers splits using a random subset of predictors each time (# is a tuning parameter)

But why?

- If a really strong predictor exists, every bootstrap tree will probably use it for the first split (2nd split, etc.)
- Makes bagged trees predictions more correlated
- Correlation --> smaller reduction in variance from aggregation

Using `tidymodels` to Fit a Random Forest Model

By randomly selecting a subset of predictors, a good predictor or two won't dominate the tree fits

- Rules of thumb exist for the number to use but better to use CV!
- Let's use the same recipe but fit a random forest model

```
rf_spec <- rand_forest(mtry = tune()) |>
  set_engine("ranger") |>
  set_mode("classification")
```

Using `tidymodels` to Fit a Random Forest Model

- Create our workflows

```
rf_wkf <- workflow() |>
  add_recipe(LR3_rec) |>
  add_model(rf_spec)
```

Using `tidymodels` to Fit a Random Forest Model

- Fit to our CV folds!
    - Note: CV isn't really necessary. We could use out-of-bag observations to determine how well our model works instead!

```
rf_fit <- rf_wkf |>
  tune_grid(resamples = heart_CV_folds,
            grid = 7,
            metrics = metric_set(accuracy, mn_log_loss))
```

Using `tidymodels` to Fit a Random Forest Model

- Check our metrics across the folds!
- Look at log loss and sort it

```
rf_fit |>
  collect_metrics() |>
  filter(.metric == "mn_log_loss") |>
  arrange(mean)
```

```
## # A tibble: 7 x 7
##    mtry .metric     .estimator  mean     n std_err .config
##   <int> <chr>       <chr>      <dbl> <int>   <dbl> <chr>
## 1     3 mn_log_loss binary     0.452    10 0.0197  Preprocessor1_Model4
## 2     4 mn_log_loss binary     0.461    10 0.0229  Preprocessor1_Model6
## 3     6 mn_log_loss binary     0.475    10 0.0257  Preprocessor1_Model7
## 4     7 mn_log_loss binary     0.480    10 0.0265  Preprocessor1_Model1
## 5     1 mn_log_loss binary     0.500    10 0.00983 Preprocessor1_Model2
## 6     8 mn_log_loss binary     0.527    10 0.0491  Preprocessor1_Model3
## 7    10 mn_log_loss binary     0.533    10 0.0489  Preprocessor1_Model5
```

Using `tidymodels` to Fit a Random Forest Model

- Get our best tuning parameter

```
rf_best_params <- select_best(rf_fit, metric = "mn_log_loss")
rf_best_params
```

```
## # A tibble: 1 x 2
##    mtry .config
##   <int> <chr>
## 1     3 Preprocessor1_Model4
```

- Refit on the entire training set using this tuning parameter

```
rf_final_wkf <- rf_wkf |>
  finalize_workflow(rf_best_params)
rf_final_fit <- rf_final_wkf |>
  last_fit(heart_split, metrics = metric_set(accuracy, mn_log_loss))
```

Compare our Models on the Test Set

- Random Forest model does better than bagging! Could tune more parameters to possibly improve

```
LR_final_fit |> collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.826 Preprocessor1_Model1
## 2 mn_log_loss binary         0.442 Preprocessor1_Model1
```

```
bag_final_fit |> collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.810 Preprocessor1_Model1
## 2 mn_log_loss binary         0.495 Preprocessor1_Model1
```

```
rf_final_fit |> collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric     .estimator .estimate .config
##   <chr>       <chr>          <dbl> <chr>
## 1 accuracy    binary         0.799 Preprocessor1_Model1
## 2 mn_log_loss binary         0.473 Preprocessor1_Model1
```

Recap

Averaging many trees can greatly improve prediction

- Comes at a loss of interpretability
- Variable importance measures can be used

Bagging

- Fit many trees on bootstrap samples and combine predictions in some way

Random Forest

- Do bagging but randomly select the predictors to use at each split